Tema

X

ADQUISICIÓN Y TRATAMIENTO DE DATOS

Departamento de Ciencias de la Computación e IA

Subprogramas en C

Objetivo

Una vez que tengamos un programa que resuelva un problema concreto, ser capaces de usarlo tantas veces como queramos sin tener que reescribirlo.

En C:


```
#include ...

// Programa principal

int main (int argc, char **argv)
{
  ...
}

// Subprogramas

...     // Funciones y procedimientos
```

IMPORTANTE: Los subprogramas sólo se ejecutan cuando son invocados desde el programa principal o desde otros subprogramas.

Introducción

Para crear un subprograma, lo primero que tenemos que hacer es escribir su cabecera (prototipo), que incluye el nombre del subprograma y su lista de parámetros:


```
float sqrt (float x)
...
```

Además de la cabecera, tendremos que escribir el cuerpo de la función:

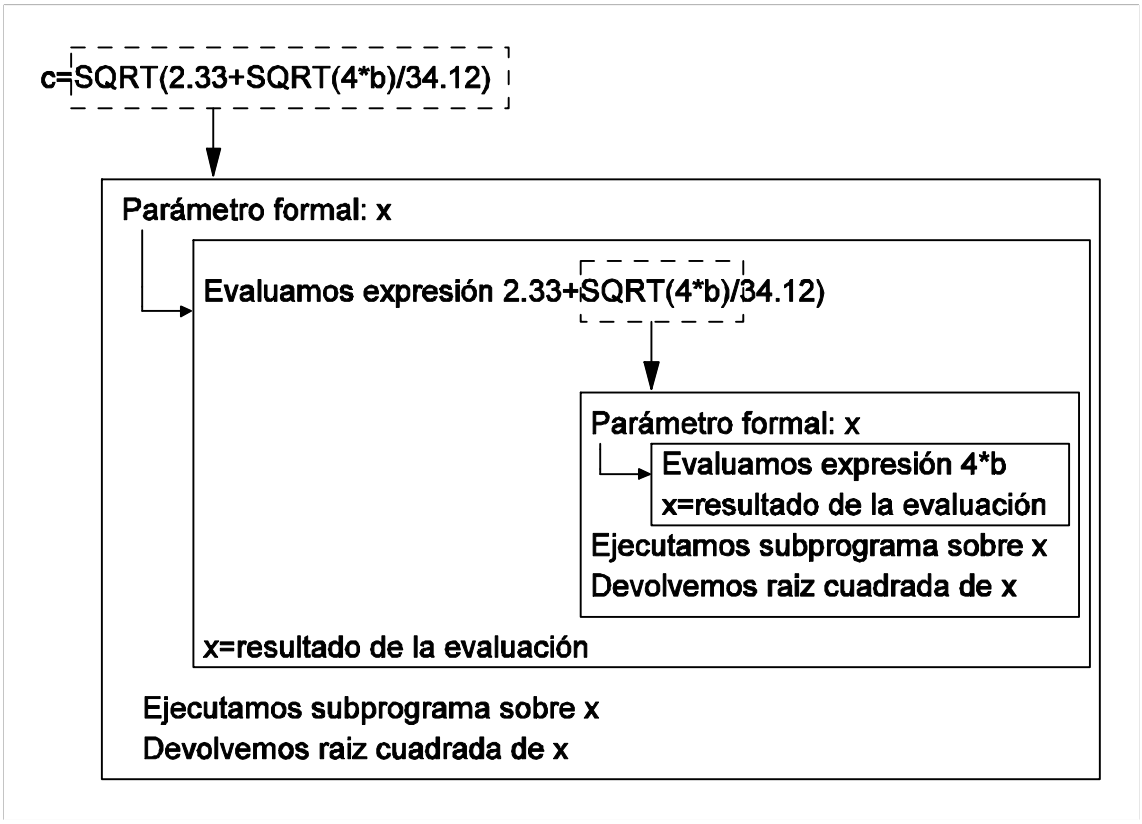

```
...
// Calculamos la raíz cuadrada
...
// Valor devuelto por la función
return ...
```

Una vez que hemos completado la implementación del subprograma, podemos usarlo cuando lo necesitemos:

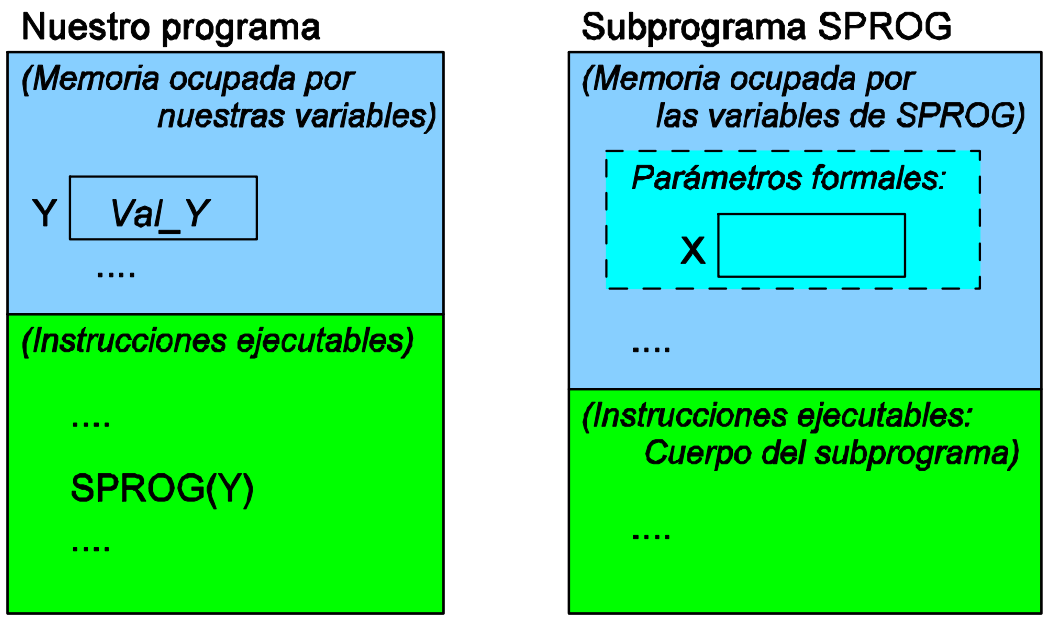

```
a = sqrt(4.0)
b = 2*sqrt(a+delta)
c = 1 / sqrt(b)
x = (-b + sqrt(b*b-4*a*c))/(2*a)
```

Parámetros formales y parámetros actuales

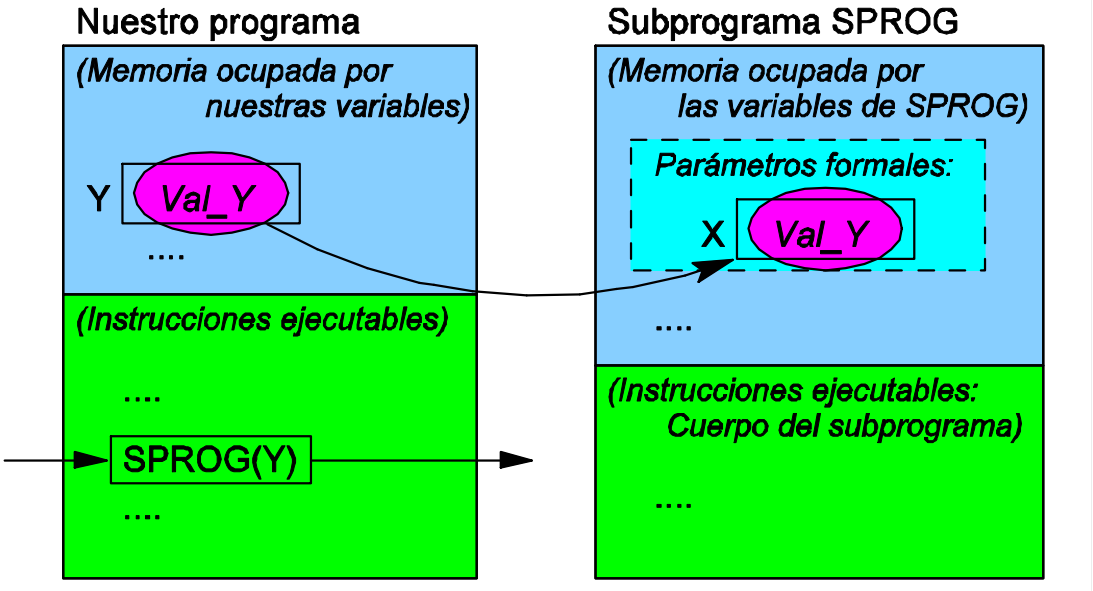
```
c = sqrt(2.33+sqrt(4*b)/34.12)
```



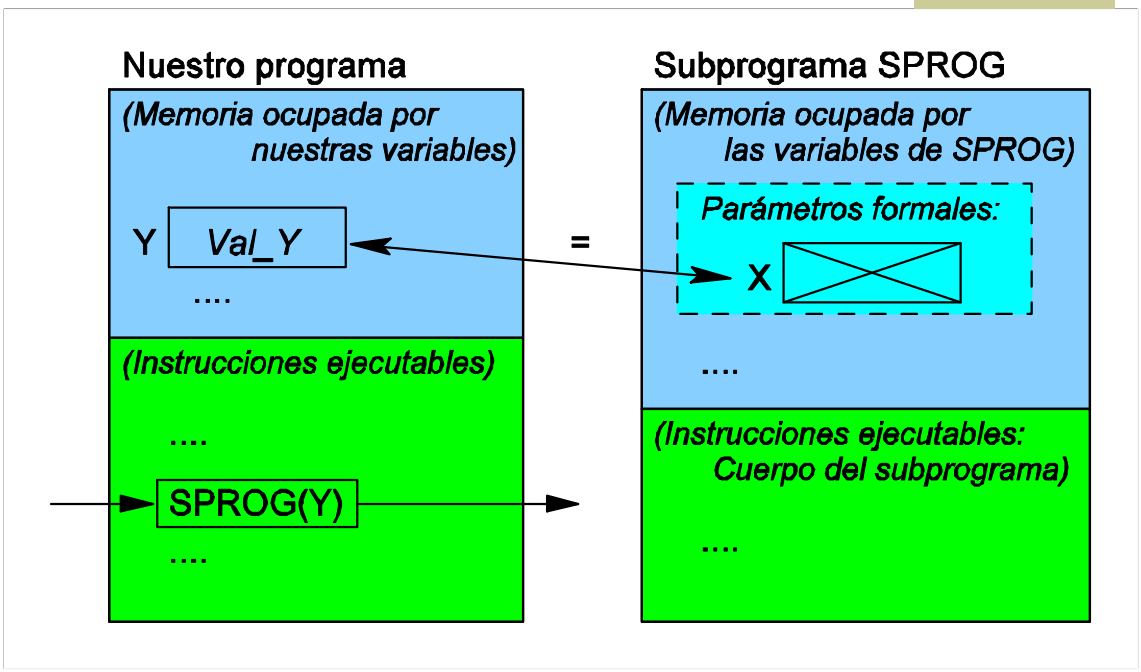
Paso de parámetros por valor



El subprograma trabaja sobre una copia del valor del parámetro actual:

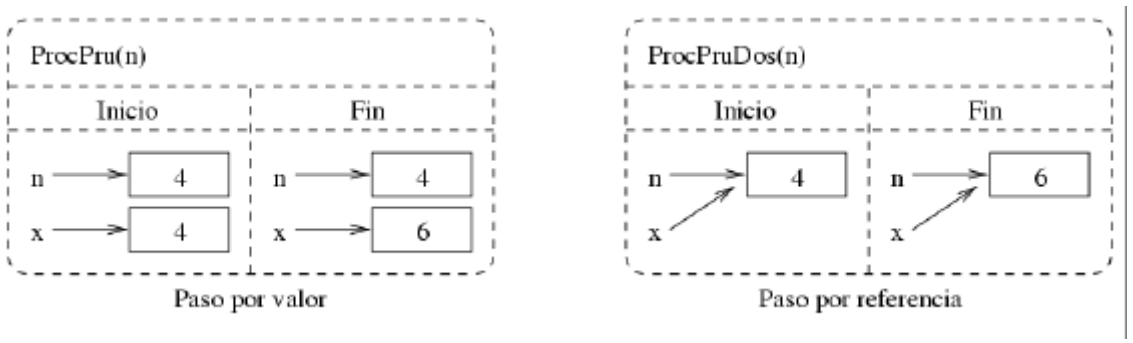


Paso de parámetros por referencia



Todos los cambios que hagamos dentro del subprograma para modificar un parámetro pasado por referencia se reflejarán en el parámetro actual correspondiente.

PASO POR VALOR vs. PASO POR REFERENCIA



NOTA

En ANSI C sólo existe el paso de parámetros por valor. El paso de parámetros por referencia lo simulamos utilizando punteros.

Funciones

Una función es un subprograma que devuelve, de forma explícita, un valor:


```
tipo nombre (parámetros formales)
{
  ! Declaraciones locales
  ...
  ! Sentencias ejecutables
  ...
  ! Valor devuelto
  return ...;
}
```

Ejemplo: Programa modular para sumar dos números


```
int suma (int a, int b)
{
  return a+b;
}

int main ()
{
  int x,y,s;

  printf(“Escriba 2 números enteros");
  scanf(“%d %d”, &x, &y);

  s = suma(x,y);
  printf(“La suma es %d", s);

  return 0;
}
```

Procedimientos

A diferencia de las funciones, no devuelven ningún valor:


```
void nombre (parámetros formales)
{
  // Declaraciones locales
  ...
  // Sentencias ejecutables
  ...
}
```

Llamada a la subrutina:


```
nombre (parámetros actuales)
```

Ejemplo: Visualizar un mensaje en pantalla

Declaración:

```
void mostrarMensaje(char *mensaje);
```

Uso:

```
mostrarMensaje("Mensaje informativo");
```

Implementación:

```
void mostrarMensaje (char *mensaje)
{
  printf("MiPrograma v1.0\n");
  printf("%s\n", mensaje);
}
```

Paso de parámetros en C

En C sólo existe el paso de parámetros por valor:


```
// Paso de parámetros por valor


void sub (int x, int y)
{
  x = x+y;
}


int main ()
{
  int z,t;

  z=5;
  t=10;

  printf("%d %d\n",z,t); // 5 10

  sub(z,t);
  printf("%d %d\n",z,t); // 5 10

  sub(z,4);
  printf("%d %d\n",z,t); // 5 10

  sub(z,2*t+20);
  printf("%d %d\n",z,t); // 5 10

  sub(3,20);
  printf("%d %d\n",z,t); // 5 10

  sub(z+32,20);
  printf("%d %d\n",z,t); // 5 10
}
```

El paso de atributos por referencia se puede simular si utilizamos punteros (pasamos como parámetros las direcciones en memoria de las variables que queremos modificar dentro del subprograma):


```
// Paso de parámetros por referencia


void sub (int *x, int y)
{
  *x = (*x) + y;
}


int main ()
{
  int z,t;

  z=5;
  t=10;

  printf("%d %d\n",z,t); // 5 10

  sub(&z,t);
  printf("%d %d\n",z,t); // 15 10

  sub(&z,4);
  printf("%d %d\n",z,t); // 19 10

  sub(&z,2*t+20);
  printf("%d %d\n",z,t); // 59 10


  sub(&3,20);            // ERROR

  sub(&(z+32),20);       // ERROR

}
```

Ámbito de las variables

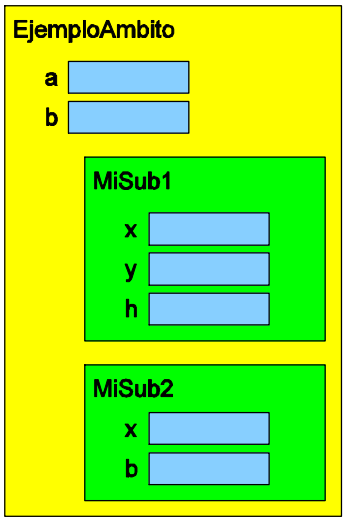
```
#include <stdio.h>

int a,b;

void MiSub1 (int x, int *y)
{
  int h = x*4;
  *y = h + x + 3 + a;
}

void MiSub2 (int x)
{
  int b = 2*x;
  a = b + 3;
}

int main ()
{
  a=3;
  b=4;
  MiSub1(a,&b);
  printf("%d %d",a,b);
  MiSub2(32);
  printf("%d %d",a,b);
}
```



Salida del programa:

```
3  21

67  21
```

Se desaconseja el uso de variables globales

Si un subprograma requiere utilizar un dato externo, dicho dato ha de pasársele como parámetro al subprograma (para que el subprograma conserve su "independencia")

Variables estáticas

`static`
Variables locales de tipo estático

```
void ejemplo ()
{
  int x = 3;          // = auto int x=3;
  static int y = 4;
  ...
}
```

La primera vez que ejecutemos el subprograma, se le asignará el valor 3 a la variable ‘x’ y el valor 4 a la variable ‘y’. A partir de ese momento, NO se volverá a asignar a ‘y’ el valor 4, sino que se recordará cuál era el último valor de ‘y’ en la ejecución anterior del subprograma.

Ejercicio: ¿Salida del siguiente programa?


```
#include <stdio.h>

void miSub ()
{
  static int x = 4;

  printf(“%d\n”, x);
  x++;
}

int main ()
{
  miSub();
  miSub();
  miSub();
}
```

Ejemplo: Intercambio de dos variables enteras


```
void swap (int *x, int *y)
{
  int tmp;

  tmp = *x;
  *x = *y;
  *y = tmp;
}
```

Uso: `swap(&a,&b);`

Ejemplo: Distancia euclídea entre dos puntos en el plano


```
#include <stdio.h>
#include <math.h>

typedef  struct Punto2D {
  float x,y;
} Punto2D;

float distancia (Punto2D p, Punto2D q)
{
  return sqrt( (p.x-q.x)*(p.x-q.x)
              +(p.y-q.y)*(p.y-q.y));
}

int main (int argc, char *argv[])
{
  Punto2D p1, p2;

  p1.x = 1; p1.y = 1;
  p2.x = 0; p2.y = 0;

  printf("Distancia: %f\n", distancia(p1,p2));
  return 0;
}
```

Ejercicios:
- Captura perfecta de datos
- Las siete y media